REMARKS/ARGUMENTS

Interview After Final

Today, the Examiner granted an interview to discuss the meaning of "immediate value" and the 35 U.S.C. §112, second paragraph, rejection. No agreement was sought nor was any agreement reached as this was just a courtesy call to expedite prosecution.

Applicant notes the Examiner returned my phone call within hours of my leaving a message. As interviews after final are discretionary, the Applicant especially appreciates this opportunity and the prompt attention given.

35 U.S.C. §112 Rejection

Claims 1-2, 4-8 and 23 are rejected under 35 U.S.C. §112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter. There is some confusion regarding the clause in claim 1, "the *immediate* value is specified in an instruction that identifies the first source register." An "immediate value" is a constant that appears in the instruction itself and does not require access to the register file to determine the value of the constant. The immediate form of instruction is discussed in the application in the two paragraphs that begin at the bottom of pages 8 and 9. Also, three references are provided that shed light on what the term "immediate value" means in the relevant art. More specifically, the following three references are provided:

1. ROSENBERG, JERRY M., *Dictionary of Computers, Information Processing & Telecommunications*, pp. 282, Second Edition, 1987 John Wiley & Sons, Inc.;

2. Intel® XScale™ Microarchitecture Assembly Language Quick Reference Card, 4 pages, Intel Corporation 2001, USA; and

3. *Machine instruction-encyclopedia article about Machine instruction*, Article on Wikipedia.org, downloaded 07/19/2004 http://encyclopedia.thefreedictionary.com/Machine%20instruction.

Once the term "immediate value" is understood, it can be seen that the language added to claims 1, 9 and 18 merely expresses the definition that is well known in the art.

Returning to clause in question, the term "that" is a defining or restrictive pronoun. Accordingly, the clause "that identifies the first source register" would modify the immediately proceeding noun, more specifically, "instruction." Accordingly, grammar dictates that the referenced portion of the claim to mean that the instruction specifies both the immediate value and the first source register. If this is still unclear, an amendment can be made to comport with the above explanation. Reconsideration of the rejection is respectfully requested.

Drawing Objection

Applicant notes ample showing of the definition added to the claim. More specifically, immediate values are referenced in the application as simm9 or simm13, and are shown in FIG. 4, items 412 and 432; FIG. 5, simm9; FIGs. 7, 8A and 8b, item 700; and FIG. 9, step 904. No change to the drawings is believed necessary to comply with 37 CFR 1.83(a) given ample showing in the current drawings.

Various 35 U.S.C. §102 & §103 Rejections.

The claims are variously rejected in four separate rejections. First, the Office Action has rejected claims 1-7, 18 and 21-22 under 35 U.S.C. §102(b) as being anticipated by the cited portions of U.S. Patent No. 5,959,874 to Lin et al. (hereinafter "Lin"). Secondly, the Office Action has rejected claims 8 and 19-20 under 35 U.S.C. §103(a) as being obvious over Lin. Also, the Office Action has rejected claims 9-17 under 35 U.S.C. §103(a) as being obvious over Lin in view of the cited portions of U.S. Patent No. 6,243,730 to Wang (hereinafter "Wang"). Finally, the Office Action has rejected claims 1 and 18 under 35 U.S.C. §103(a) as being obvious over Lin in view of the cited portions of U.S. Patent No. 5,831,885 to Mennemeier (hereinafter "Mennemeier").

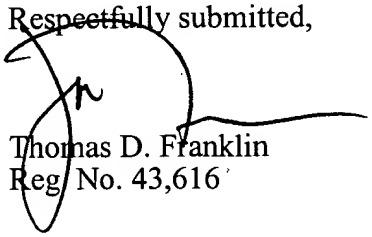
All these rejections rely upon Lin for a showing of an immediate value, but as described in the last response, Lin does not contemplate the immediate form of instructions. A source register (SRC1 or SRC2) is always specified in Lin. Indeed, the term "immediate" does not even appear in the Lin reference. Since all the rejections in the prior paragraph hinge on Lin teaching this element, Applicants respectfully request reconsideration of the rejection.

CONCLUSION

In view of the foregoing, Applicants believe all claims now pending in this Application are in condition for allowance and an action to that end is respectfully requested.

If the Examiner believes a telephone conference would expedite prosecution of this application, please telephone the undersigned at 303-571-4000.

Respectfully submitted,



Thomas D. Franklin
Reg. No. 43,616

TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, Eighth Floor
San Francisco, California 94111-3834
Tel: 303-571-4000
Fax: 415-576-0300
TDF:cmb

Enclosures (3 references)

60227326 v1

DICTIONARY OF

COMPUTERS, INFORMATION PROCESSING & TELECOMMUNICATIONS

SECOND EDITION

automatic sequencing: the ability of equipment to put information in order or in a connected series without human intervention.

impulse noise: short bursts of high-level noise such as that resulting from the coupling of transients into a channel. Typical sources of such noises are lightning and transients from switching systems. Impulse noise, which sounds like a click, is not particularly detrimental to voice communications, but it can be detrimental to data communications. Some of the older switching systems, such as the Panel type, create so much impulse noise that DATAPHONE service is not handled by central offices of this type.

miscellaneous common carrier (MCC): a communications common carrier which is not engaged in the business of providing either a public landline message telephone service or public message telegraph service. Miscellaneous common carriers were initially authorized to serve TV and radio markets. Today they are still viewed as serving these markets, although Domestic Satellite Carriers and Specialized Common Carriers also meet the FCC definition of Miscellaneous Common Carriers.

portability: the ability to use data sets or files with differing operating systems. Volumes whose data sets or files are cataloged in a user catalog can be demounted from storage devices of one system, moved to another system and mounted on storage devices of that system.

remote call forwarding (RCF): a service offering which allows customers to have a telephone number in an ESS office without having any other local telephone service in that office. Calls coming to the remote call forwarding number are automatically forwarded to any answering location the customer wants.

zero suppression: the elimination from a numeral of zeros that have no significance in the numeral. Zeros that have no significance include those to the left of the nonzero digits in the integral part of a numeral and those to the right of the nonzero digits in the fractional part. Or on a calculator, the process by which unwanted zeros are omitted from the printed or displayed result of a calculation.

images without the use of coordinate data, for example, form flash. synonymous with *noncoded graphics*. (E)

image printer: a printer using optical technology to compose an image of a complete page from digital input.

image processing: processing of images using computer techniques.

image sensor: synonymous with *charge couple device*.

image space: see *display space, image storage space*.

image storage space: in computer graphics, the storage locations occupied by a coded image. synonymous with *coded image space*. (E)

IML (IMPL): initial microcode (or microprogram) load. The technique for loading a microprogram into main memory, usually from a diskette.

immediate access: the ability of a computer to enter and retrieve data from memory without delay.

immediate-access storage: a storage device whose access time is negligible in comparison with other operating times. (A)

immediate address: the contents of an address part that contains the value of an operand rather than an address. synonymous with *zero-level address*. *(A) (B)*

immediate addressing: a method of addressing in which the address part of an instruction contains an immediate address. (A) (B)

immediate data

(1) data contained in an instruction rather than in a separate storage location.

(2) data transferred during instruction execution time.

immediate instruction: an instruction that contains within itself an operand for the operation specified, rather than an address of the operand. (A) (B)

immediate-mode commands: system and editing commands that are executed as soon as the carriage control key (RETURN, ENTER) is pressed.

immediate processing: synonymous with *demand processing*.

immediate task: a task assigned the second highest-level dispatching priority by the network control program. It initiates I/O operations on lines that are in an idle state. see also *appendate task, nonproductive task, productive task*.

impact paper: a coated paper that may be used to get one or more copies of printed, typed, or handwritten information without the need for a ribbon or other inking device. Each sheet is coated on the front. Pressure on the top of the top sheet causes the character to appear on the front of that sheet and on the front of subsequent sheets underneath the top sheet, thus eliminating the need for carbon paper between sheets.

impact printer (IP): a printer in which printing is the result of mechanical impacts. (A) (B)

impedance: the combined effect of resistance, inductance, and capacitance on a signal at a particular frequency.

impedance compensator: in a transmission path, the network that obtains the desired characteristic over a particular frequency range.

imperative macroinstruction: macroinstructions that are converted into object program instructions.

imperative operation: an instruction requiring the manipulating of data by the computer.

imperative statement: a statement that specifies an action to be taken unconditionally. (E) synonym for *instruction*. (B)

implement: carrying out or giving physical, functional reality to a plan. Some software are tools for imple-

Intel® XScale™ Microarchitecture Assembly Language Quick Reference Card
ARM Instruction Set

Operation		Assembler	S updates	Action	Notes
Move	Move	MOV{cond}{S} Rd, <Oprnd2>	N Z C	Rd := Oprnd2	
	NOT	MVN{cond}{S} Rd, <Oprnd2>	N Z C	Rd := 0xFFFFFFFF EOR Oprnd2	
	SPSR to register	MRS{cond} Rd, SPSR		Rd := SPSR	
	CPSR to register	MRS{cond} Rd, CPSR		Rd := CPSR	
	register to SPSR	MSR{cond} SPSR_<fields>, Rm		SPSR := Rm (selected bytes only)	
	register to CPSR	MSR{cond} CPSR_<fields>, Rm		CPSR := Rm (selected bytes only)	
	immediate to SPSR	MSR{cond} SPSR_<fields>, #<immed_8r>		SPSR := immed_8r (selected bytes only)	
	immediate to CPSR	MSR{cond} CPSR_<fields>, #<immed_8r>		CPSR := immed_8r (selected bytes only)	
Arithmetic	Add	ADD{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn + Oprnd2	
	with carry	ADC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn + Oprnd2 + Carry	
	saturating	QADD{cond} Rd, Rm, Rn		Rd := SAT(Rm + Rn)	Sticky. No shift/rotate.
	double saturating	QDADD{cond} Rd, Rm, Rn		Rd := SAT(Rm + SAT(Rn * 2))	Sticky. No shift/rotate.
	Subtract	SUB{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn - Oprnd2	
	with carry	SBC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn - Oprnd2 - NOT(Carry)	
	reverse subtract	RSB{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Oprnd2 - Rn	
	reverse subtract with carry	RSC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Oprnd2 - Rn - NOT(Carry)	
	saturating	QSUB{cond} Rd, Rm, Rn		Rd := SAT(Rm - Rn)	Sticky. No shift/rotate.
	double saturating	QDSUB{cond} Rd, Rm, Rn		Rd := SAT(Rm - SAT(Rn * 2))	Sticky. No shift/rotate.
	Multiply	MUL{cond}{S} Rd, Rm, Rs	N Z C	Rd := (Rm * Rs)[31:0]	
	accumulate	MLA{cond}{S} Rd, Rm, Rs, Rn	N Z C	Rd := ((Rm * Rs) + Rn)[31:0]	
	unsigned long	UMULL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C V	RdHi,RdLo := unsigned(Rm * Rs)	
	unsigned accumulate long	UMLAL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C V	RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs)	
	signed long	SMULL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C V	RdHi,RdLo := signed(Rm * Rs)	
	signed accumulate long	SMLAL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C V	RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs)	
	signed 16 * 16 bit	SMULxy{cond} Rd, Rm, Rs		Rd := Rm[x] * Rs[y]	No shift/rotate.
	signed 32 * 16 bit	SMULWy{cond} Rd, Rm, Rs		Rd := (Rm * Rs[y])[47:16]	No shift/rotate.
	signed accumulate 16 * 16	SMLAxy{cond} Rd, Rm, Rs, Rn		Rd := Rn + Rm[x] * Rs[y] Sticky.	No shift/rotate.
	signed accumulate 32 * 16	SMLAWy{cond} Rd, Rm, Rs, Rn		Rd := Rn + (Rm * Rs[y])[47:16]	Sticky. No shift/rotate.
	signed accumulate long 16 * 16	SMLALxy{cond} RdLo, RdHi, Rm, Rs		RdHi,RdLo := RdHi,RdLo + Rm[x] * Rs[y]	No shift/rotate.
	Count leading zeroes	CLZ{cond} Rd, Rm		Rd := number of leading zeroes in Rm	
DSP CP0	Multiply with internal accumulate	MIA{cond} acc0, Rm, Rs		acc0 = (Rm[31:0] * Rs[31:0])[39:0] + acc0[39:0])	
		MIAPH{cond} acc0, Rm, Rs		acc0 = sign_extend(Rm[31:16] * Rs[31:16]) + sign_extend(Rm[15:0] * Rs[15:0]) + acc0[39:0]	
		MIAxy{cond} acc0, Rm, Rs		acc0[39:0] = sign_extend(Rm[x] * Rs[y]) + acc0[39:0]	
	Accumulator move to	MAR{cond} acc0, RdLo, RdHi		acc0[39:32] = RdHi[7:0] acc0[31:0] = RdLo[31:0]	
	Accumulator move from	MRA{cond} RdLo, RdHi, acc0		RdHi[31:0] = sign_extend(acc0[39:32] RdLo[31:0] = acc0[31:0]	
Logical	Test	TST{cond} Rn, <Oprnd2>	N Z C	Update CPSR flags on Rn AND Oprnd2	
	Test equivalence	TEQ{cond} Rn, <Oprnd2>	N Z C	Update CPSR flags on Rn EOR Oprnd2	
	AND	AND{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn AND Oprnd2	
	EOR	EOR{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn EOR Oprnd2	
	ORR	ORR{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn OR Oprnd2	
	Bit Clear	BIC{cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn AND NOT Oprnd2	
	No operation	NOP		R0 := R0	Flags not affected.
	Shift/Rotate				See Table Operand 2.
Compare	Compare	CMP{cond} Rn, <Oprnd2>	N Z C V	Update CPSR flags on Rn - Oprnd2	
	negative	CMN{cond} Rn, <Oprnd2>	N Z C V	Update CPSR flags on Rn + Oprnd2	
Branch	Branch	B{cond} label		R15 := label	label within ±32Mb
	with link	BL{cond} label		R14 := R15-4, R15 := label	label within ±32Mb
	and exchange	BX{cond} Rm		R15 := Rm, Change to Thumb if Rm[0] is 1	
	with link and exchange (1)	BLX label		R14 := R15 - 4, R15 := label	Cannot be conditional.
	with link and exchange (2)	BLX{cond}		Rm R14 := R15 - 4, R15 := Rm[31:1]	
Load	Word	LDR{cond} Rd, <a_mode2>		Rd := [address]	
	User mode privilege	LDR{cond}T Rd, <a_mode2P>			
	branch (and exchange)	LDR{cond} R15, <a_mode2>		R15 := [address][31:1], Change to Thumb if [address][0] is 1	
	Byte	LDR{cond}B Rd, <a_mode2>		Rd := ZeroExtend[byte from address]	
	User mode privilege	LDR{cond}BT Rd, <a_mode2P>			
	signed	LDR{cond}SB Rd, <a_mode3>		Rd := SignExtend[byte from address]	
	Halfword	LDR{cond}H Rd, <a_mode3>		Rd := ZeroExtent[halfword from address]	
	signed	LDR{cond}SH Rd, <a_mode3>		Rd := SignExtend[halfword from address]	
Load Multiple	Pop, or Block data load	LDM{cond}<a_mode4L> Rd{!}, <reglist-pc>		Load list of registers from [Rd]	
	return (and exchange)	LDM{cond}<a_mode4L> Rd{!}, <reglist+pc>		Load registers, R15 := [address][31:1]	
	and restore CPSR	LDM{cond}<a_mode4L> Rd{!}, <reglist+pc>^		Load registers, branch and exchange	Exception modes only.
	User mode registers	LDM{cond}<a_mode4L> Rd, <reglist-pc>^		Load list of User mode registers from [Rd]	Privileged modes only.
Load Double		LDRD			
Store	Word	STR{cond} Rd, <a_mode2>		[address] := R	
	User mode privilege	STR{cond}T Rd, <a_mode2P>		[address] := Rd	
	Byte	STR{cond}B Rd, <a_mode2>		[address][7:0] := Rd[7:0]	
	User mode privilege	STR{cond}BT Rd, <a_mode2P>		[address][7:0] := Rd[7:0	
	Halfword	STR{cond}H Rd, <a_mode3>		[address][15:0] := Rd[15:0]	
Store Multiple	Push, or Block data store	STM{cond}<a_mode4S>Rd{!}, <reglist>		Store list of registers to [Rd]	
	User mode registers	STM{cond}<a_mode4S> Rd{!}, <reglist>^		Store list of User mode registers to [Rd]	Privileged modes only.
Store Double		STRD			
Swap	Word	SWP{cond} Rd, Rm, [Rn]		temp := [Rn], [Rn] := Rm, Rd := temp	
	Byte	SWP{cond}B Rd, Rm, [Rn]		temp := ZeroExtend([Rn][7:0]), [Rn][7:0] := Rm[7:0], Rd := temp	
Coprocessor	Move to ARM reg from coproc	MRC{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>			
		MRRC{cond} p<cpnum>, <op1>, <Rd>, <Rn>, <CRn>			
	Move to coproc from ARM reg	MCR{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>			
		MCRR{cond} p<cpnum>, <op1>, <Rd>, <Rn>, <CRn>			
	Load coprocessor	LDC{cond} p<cpnum>, CRd, <a_mode5>		CRd := [address]	
	Store coprocessor	STC{cond} p<cpnum>, CRd, <a_mode5>		[address] := CRd	
Software interrupt		SWI{cond} <immed_24>		Software interrupt processor exception	24-bit value.
Breakpoint		BKPT <immed_16>		Prefetch abort or enter debug state	Cannot be conditional.
Pre-load		PLD <a_mode2>		Pre-load cache line	

Intel® XScale™ Microarchitecture Assembly Language Quick Reference Card
Thumb Instruction Set

Operation		Assembler	Action	Notes
Move	Immediate	MOV Rd, #<immed_8>	Rd := immed_8	8-bit immediate value.
	Lo to Lo	MOV Rd, Rm	Rd := Rm	
	Hi to Lo, Lo to Hi, Hi to Hi	MOV Rd, Rm	Rd := Rm	Not Lo to Lo
Arithmetic	Add	ADD Rd, Rn, #<immed_3>	Rd := Rn + immed_3	3-bit immediate value.
	Lo and Lo	ADD Rd, Rn, Rm	Rd := Rn + Rm	
	Hi to Lo, Lo to Hi, Hi to Hi	ADD Rd, Rm	Rd := Rd + Rm	Not Lo to Lo
	immediate	ADD Rd, #<immed_8>	Rd := Rd + immed_8	8-bit immediate value.
	with carry	ADC Rd, Rm	Rd := Rd + Rm + C-bit	
	value to SP	ADD SP, #<immed_7*4>	SP := SP + immed_7 * 4	9-bit immediate value (word-aligned).
	form address from SP	ADD Rd, SP, #<immed_8*4>	Rd := SP + immed_8 * 4	10-bit immediate value (word-aligned).
	form address from PC	ADD Rd, PC, #<immed_8*4>	Rd := (PC AND 0xFFFFFFFC) + immed_8 * 4	10 bit immediate value (word-aligned).
	Subtract	SUB Rd, Rn, Rm	Rd := Rn - Rm	
	immediate 3	SUB Rd, Rn, #<immed_3>	Rd := Rn - immed_3	3-bit immediate value.
	immediate 8	SUB Rd, #<immed_8>	Rd := Rd - immed_8	8-bit immediate value.
	with carry	SBC Rd, Rm	Rd := Rd - Rm - NOT C-bit	
	value from SP	SUB SP, #<immed_7*4>	SP := SP - immed_7 * 4	9-bit immediate value (word-aligned).
	Negate	NEG Rd, Rm	Rd := - Rm	
	Multiply	MUL Rd, Rm	Rd := Rm * Rd	
	Compare	CMP Rn, Rm	update CPSR flags on Rn - Rm	Can be Lo to Lo, Lo to Hi, Hi to Lo, or Hi to Hi.
	negative	CMN Rn, Rm	update CPSR flags on Rn + Rm	
	immediate	CMP Rn, #<immed_8>	update CPSR flags on Rn - immed_8	8-bit immediate value.
	No operation	NOP	R8 := R8	lags not affected.
Logical	AND	AND Rd, Rm	Rd := Rd AND Rm	
	Exclusive OR	EOR Rd, Rm	Rd := Rd EOR Rm	
	OR	ORR Rd, Rm	Rd := Rd OR Rm	
	Bit clear	BIC Rd, Rm	Rd := Rd AND NOT Rm	
	Move NOT	MVN Rd, Rm	Rd := NOT Rm	
	Test bits	TST Rn, Rm	update CPSR flags on Rn AND Rm	
Shift / Rotate	Logical shift left	LSL Rd, Rm, #<immed_5>	Rd := Rm << immed_5	5-bit immediate shift. Allowed shifts 0-31.
	LSL Rd, Rs	Rd := Rd << Rs		
	Logical shift right	LSR Rd, Rm, #<immed_5>	Rd := Rm >> immed_5	5-bit immediate shift. Allowed shifts 1-32.
	LSR Rd, Rs	Rd := Rd >> Rs		
	Arithmetic shift right	ASR Rd, Rm, #<immed_5>	Rd := Rm ASR immed_5	5-bit immediate shift. Allowed shifts 1-32.
	ASR Rd, Rs	Rd := Rd ASR Rs		
	Rotate right	ROR Rd, Rs	Rd := Rd ROR Rs	
Branch	Conditional branch	B{cond} label	R15 := label	label must be within -252 to +258 bytes
		See Table Condition Field (ARM side).		
		AL not allowed.		
	Unconditional branch	B label	R15 := label	label must be within ±2Kb of current instruction.
	Long branch with link	BL label	R14 := R15 - 2, R15 := label	Encoded as two Thumb instructions.
		label must be within ±4Mb of current instruction.		
	Branch and exchange	BX Rm	R15 := Rm AND 0xFFFFFFFE	Change to ARM state if Rm[0] = 0.
	Branch with link and exchange	BLX label	R14 := R15 - 2, R15 := label	Encoded as two Thumb instructions.
	Change to ARM	label must be within ±4Mb of current instruction.		
	Branch with link and exchange	BLX Rm	R14 := R15 - 2, R15 := Rm AND 0xFFFFFFFE	
	Change to ARM if Rm[0] = 0			
Software Interrupt		<immed_8>	Software interrupt processor exception	8-bit immediate value encoded in instruction.
Breakpoint		BKPT <immed_8>	Prefetch abort or enter debug state	
Load	with immediate offset, word	LDR Rd, [Rn, #<immed_5*4>]	Rd := [Rn + immed_5 * 4]	
	halfword	LDRH Rd, [Rn, #<immed_5*2>]	Rd := ZeroExtend([Rn + immed_5 * 2][15:0])	Clears bits 31:16
	byte	LDRB Rd, [Rn, #<immed_5>]	Rd := ZeroExtend([Rn + immed_5][7:0])	Clears bits 31:8
	with register offset, word	LDR Rd, [Rn, Rm]	Rd := [Rn + Rm]	
	halfword	LDRH Rd, [Rn, Rm]	Rd := ZeroExtend([Rn + Rm][15:0])	Clears bits 31:16
	signed halfword	LDRSH Rd, [Rn, Rm]	Rd := SignExtend([Rn + Rm][15:0])	Sets bits 31:16 to bit 15
	byte	LDRB Rd, [Rn, Rm]	Rd := ZeroExtend([Rn + Rm][7:0])	Clears bits 31:8
	signed byte	LDRSB Rd, [Rn, Rm]	Rd := SignExtend([Rn + Rm][7:0])	Sets bits 31:8 to bit 7
	PC-relative	LDR Rd, [PC, #<immed_8*4>]	Rd := [(PC AND 0xFFFFFFFC) + immed_8 * 4]	
	SP-relative	LDR Rd, [SP, #<immed_8*4>]	Rd := [SP + immed_8 * 4]	
	Multiple	LDMIA Rn!, <reglist>	Loads list of registers	Always updates base register.
Store	with immediate offset, word	STR Rd, [Rn, #<immed_5*4>]	[Rn + immed_5 * 4] := Rd	
	halfword	STRH Rd, [Rn, #<immed_5*2>]	[Rn + immed_5 * 2][15:0] := Rd[15:0]	Ignores Rd[31:16]
	byte	STRB Rd, [Rn, #<immed_5>]	[Rn + immed_5][7:0] := Rd[7:0]	Ignores Rd[31:8]
	with register offset, word	STR Rd, [Rn, Rm]	[Rn + Rm] := Rd	
	halfword	STRH Rd, [Rn, Rm]	[Rn + Rm][15:0] := Rd[15:0]	Ignores Rd[31:16]
	byte S	STRB Rd, [Rn, Rm]	[Rn + Rm][7:0] := Rd[7:0]	Ignores Rd[31:8]
	SP-relative, word	STR Rd, [SP, #<immed_8*4>]	[SP + immed_8 * 4] := Rd	
	Multiple	STMIA Rn!, <reglist>	Stores list of registers	Always updates base register.
Push	Push	PUSH <reglist>	Push registers onto stack	Full descending stack.
	Push with link	PUSH <reglist, LR>	Push LR and registers on to stack	
Pop	Pop	POP <reglist>	Pop registers from stack	
	Pop and return	POP <reglist, PC>	Pop registers, branch to address loaded to PC	
	Pop and return with exchange	POP <reglist, PC>	Pop, branch, and change to ARM state if address[0] = 0	

All Thumb registers are Lo (R0-R7) except where specified. Hi registers are R8-R15.

Intel® XScale™ Microarchitecture Assembly Language Quick Reference Card
Key to Tables

CODES

{cond}	Refer to Table Condition Field {cond}
<Oprnd2>	Refer to Table Operand 2
<fields>	Refer to Table PSR fields
{S}	Updates condition flags if S present
Sticky	Sticky flag, updates on overflow (no S option), read and reset using MRS and MSR
x,y	B meaning half-register [15:0], or T meaning [31:16]
<immed_8r>	A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits
<immed_8*4>	A 10-bit constant, formed by left-shifting an 8-bit value by two bits
<a_mode2>	Refer to Table Addressing Mode 2
<a_mode2P>	Refer to Table Addressing Mode 2 (Post-indexed only)
<a_mode3>	Refer to Table Addressing Mode 3
<a_mode4L>	Refer to Table Addressing Mode 4 (Block load or Stack pop)
<a_mode5>	Refer to Table Addressing Mode 5
<reglist>	A list of registers, enclosed in braces ({ and })
{!}	Updates base register after data transfer if ! present

CONDITION FIELD {COND}

Mnemonic	Description
EQ	Equal
NE	Not equal
CS / HS	Carry Set / Unsigned higher or same
CC / LO	Carry Clear / Unsigned lower
MI	Negative
PL	Positive or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower or same
GE	Signed greater than or equal
LT	Signed less than
GT	Signed greater than
LE	Signed less than or equal
AL	Always (normally omitted)

OPERAND 2

Immediate value	#<immed_8r>	
Logical shift left immediate	Rm, LSL #<immed_5>	Allowed shifts 0-31
Logical shift right immediate	Rm, LSR #<immed_5>	Allowed shifts 1-32
Arithmetic shift right immediate	Rm, ASR #<immed_5>	Allowed shifts 1-32
Rotate right immediate	Rm, ROR #<immed_5>	Allowed shifts 1-31
Register	Rm	
Rotate right extended	Rm, RRX	
Logical shift left register	Rm, LSL Rs	
Logical shift right register	Rm, LSR Rs	
Arithmetic shift right register	Rm, ASR Rs	
Rotate right register	Rm, ROR Rs	

PSR FIELDS (USE AT LEAST ONE SUFFIX)

Suffix	Meaning	
c	Control field mask byte	PSR[7:0]
f	Flags field mask byte	PSR[31:24]
s	Status field mask byte	PSR[23:16]
x	Extension field mask byte	PSR[15:8]

ARM ADDRESSING MODES

ADDRESSING MODE 2 - WORD AND UNSIGNED BYTE DATA TRANSFER

Pre-indexed	Immediate offset	[Rn, #+/-<immed_12>]{!}	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
	Register offset	[Rn, +/-Rm]{!}	
	Scaled register offset	[Rn, +/-Rm, LSL #<immed_5>]{!}	Allowed shifts 0-31
		[Rn, +/-Rm, LSR #<immed_5>]{!}	Allowed shifts 1-32
		[Rn, +/-Rm, ASR #<immed_5>]{!}	Allowed shifts 1-32
		[Rn, +/-Rm, ROR #<immed_5>]{!}	Allowed shifts 1-31
		[Rn, +/-Rm, RRX]{!}	
Post-indexed	Immediate offset	[Rn], #+/-<immed_12>	
	Register offset	[Rn], +/-Rm DB	
	Scaled register offset	[Rn], +/-Rm, LSL #<immed_5>	Allowed shifts 0-31
		[Rn], +/-Rm, LSR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ASR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ROR #<immed_5>	Allowed shifts 1-31
		[Rn], +/-Rm, RRX	

ADDRESSING MODE 2 (POST-INDEXED ONLY)

Post-indexed	Immediate offset	[Rn], #+/-<immed_12>	
	Zero offset	[Rn]	Equivalent to [Rn],#0
	Register offset	[Rn], +/-Rm	
	Scaled register offset	[Rn], +/-Rm, LSL #<immed_5>	Allowed shifts 0-31
		[Rn], +/-Rm, LSR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ASR #<immed_5>	Allowed shifts 1-32
		[Rn], +/-Rm, ROR #<immed_5>	Allowed shifts 1-31
		[Rn], +/-Rm, RRX	

ADDRESSING MODE 3 - HALFWORD AND SIGNED BYTE DATA TRANSFER

Pre-indexed	Immediate offset	[Rn, #+/-<immed_8>]{!}	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
	Register	[Rn, +/-Rm]{!}	
Post-indexed	Immediate offset	[Rn], #+/-<immed_8>	
	Register	[Rn], +/-Rm	

ADDRESSING MODE 5 - COPROCESSOR DATA TRANSFER

Pre-indexed	Immediate offset	[Rn, #+/-<immed_8*4>]{!}	
	Zero offset	[Rn]	Equivalent to [Rn,#0]
Post-indexed	Immediate offset	[Rn], #+/-<immed_8*4>	
Unindexed	No offset	[Rn], {8-bit copro. option}	

ADDRESSING MODE 4 - MULTIPLE DATA TRANSFER

Block load	
IA	Increment After
IB	Increment Before
DA	Decrement After
DB	Decrement Before
Block store	
IA	Increment After
IB	Increment Before
DA	Decrement After
DB	Decrement Before
Stack pop	
FD	Full Descending
ED	Empty Descending
FA	Full Ascending
EA	Empty Ascending
Stack push	
EA	Empty Ascending
FA	Full Ascending
ED	Empty Descending
FD	Full Descending

Intel Access

Developer's Site	developer.intel.com
I/O Building Blocks Home Page	developer.intel.com/design/iio
Intel® XScale™ Microarchitecture	developers.intel.com/design/XScale
Other Intel Support	developer.intel.com/design/litcentr
Intel Literature Center	(800) 548-4725 7a.m. to 7 p.m. CST (U.S. and Canada) International locations please contact your local sales office.
General Information Hotline	(800) 628-8686 or (916) 356-3104 5 a.m. to 5 p.m. PST

For more information, visit the Intel Web site at: **developer.intel.com**

UNITED STATES AND CANADA
Intel Corporation
Robert Noyce Bldg.
2200 Mission College Blvd.
P.O. Box 58119
Santa Clara, CA 95052-8119
USA

EUROPE
Intel Corporation (UK) Ltd.
Pipers Way
Swindon
Wiltshire SN3 1RJ
UK

ASIA-PACIFIC
Intel Semiconductor Ltd.
32/F Two Pacific Place
88 Queensway, Central
Hong Kong, SAR

JAPAN
Intel Kabushiki Kaisha
P.O. Box 115 Tsukuba-gakuen
5-6 Tokodai, Tsukuba-shi
Ibaraki-ken 305
Japan

SOUTH AMERICA
Intel Semicondutores do Brazil
Rue Florida, 1703-2 and CJ22
CEP 04565-001 Sao Paulo-SP
Brazil

Printed in USA/0801/5K/IL11139 KG

Machine instruction

A system of codes directly understandable by a computer's CPU is termed this CPU's **native** or **machine language**. Although *machine code* may seem similar to *assembly language* they are in fact two different types of languages. *Assembly code* consists of both binary numbers and simple words whereas *machine code* is composed only of the two binary digits 0 and 1. Every CPU has its own machine language, although there is considerable overlap between some. If CPU *A* understands the full language of CPU *B* it is said that *A* is compatible with *B*. CPU *B* may not be compatible with CPU *A*, as *A* may know a few codes that *B* does not.

The "words" of a machine language are called *instructions*; each of these gives a basic command to the CPU. A program is just a long list of instructions that are *executed* by a CPU. Older processors executed instructions one after the other, but newer superscalar processors are capable of executing several instructions at once. Program flow may be influenced by special *jump* instructions that transfer execution to an instruction other than the following one. Conditional jumps are taken (execution continues at another address) or not (execution continues at the next instruction) depending on some condition.

Instructions are simply a pattern of bits -- different patterns correspond to different commands to the machine. The more readable rendition of a machine language is called assembly language.

Some languages give all their instructions the same number of bits, while the instruction length differs in others. How the patterns are organised depends largely on the specific language. Common to most is the division of an instruction into *fields*, of which one or more specify the exact operation (for example "add"). Other fields may give the type of the operands, their location, or their value directly (operands contained in an instruction are called *immediate*).

As a specific example, let us take a look at the MIPS architecture. Its instructions are always 32 bit long. The general type of instruction is given by the *op* field, the highest 6 bits. J-type and I-type instructions are fully specified by *op*. R-type instructions include an addtional field *funct* to determine the exact operation. The fields used in these types are:

```
6  5  5  5  5  6 bits
[ op | rs | rt | rd |shamt| funct] R-type
[ op | rs | rt | address/immediate] I-type
[ op | target address ] J-type
```

rs, *rt*, and *rd* indicate register operands; *shamt* gives a shift amount; and the *address* or *immediate* fields contain an operand directly.

For example adding the registers 1 and 2 and placing the result in register 6 is encoded:

```
[ op | rs | rt | rd |shamt| funct]
0 1 2 6 0 32 decimal
000000 00001 00010 00110 00000 100000 binary
```

Loading a value from the memory cell 68 cells after the one register 3 points to into register 8:

```
[ op | rs | rt | address/immediate]
35 3 8 68 decimal
100011 00011 01000 00000 00001 000100 binary
```

Jumping to the address 1025:

```
[ op | target address ]
2 1025 decimal
000010 00000 00000 00000 10000 000001 binary
```

See also

CISC, RISC, VLIW, Endianness.

Further reading

Patterson and Hennessy: Computer Organization and Design. The Hardware/Software Interface. Morgan Kaufmann Publishers. ISBN 1-55860-281-X

Some articles mentioning "Machine instruction":

ASM	Disassembler	Java Native Interface	National Assembly of Hungary	Register Transfer Language
Assembly	Disassembly	Jump instruction	Object language	ScreamTracker
DBX	FASM	Low-level programming language	Objectlanguage	SEX (computers)
Dbx debugger	Halt	MSIL	Random Access Machine	SSK

Previous	Encyclopedia Browser		Next
Machinae supremacy	Machine Elves	Machine intelligence	Machine politics
Machine	Machine embroidery	Machine language	Machine port
Machine (album)	Machine gun	Machine language monitor	Machine processing
Machine code	Machine Gun Molly	Machine learning	Machine reproduction
Machine code monitor	Machine Head	Machine Pistol	Machine tool

Full Dictionary Browser

Machination (law)	Machine code (enc.)	Machine intelligence (enc.)	machine politician
machinator	Machine code monitor (enc.)	Machine language	Machine politics (enc.)
Machine	machine cycle (comp.)	Machine language (comp.)	Machine port (enc.)
Machine (comp.)	Machine Elves (enc.)	Machine language (enc.)	Machine processing (enc.)
Machine (law)	Machine embroidery (enc.)	Machine language monitor (enc.)	machine readable
Machine (enc.)	Machine gun	Machine learning (comp.)	machine readable dictionary
Machine (album) (enc.)	Machine gun (enc.)	Machine learning (enc.)	Machine reproduction (enc.)
machine bolt	Machine Gun Molly (enc.)	machine operation	machine rifle
Machine code	machine gunner	Machine Pistol	machine screw
Machine code (comp.)	Machine Head (enc.)	Machine Pistol (enc.)	machine shop